

NINTENDO PROGRAMMER'S REFERENCE GUIDE
© COPYRIGHT 1989 ELECTRONIC ARTS. ALL RIGHTS RESERVED.
THIS DOCUMENT IS COVERED BY YOUR NON-DISCLOSURE AGREEMENT.
CONFIDENTIAL MATERIAL ENCLOSED

TABLE OF CONTENTS

INTRODUCTION	1
GENERAL DESCRIPTION	1
CPU Memory Map	2
PPU Memory Map	2
Screen Organization	2
Character Data Organization	5
Screen Colors	7
Sprite Programming	9
Sprite Colors	13
Split Screen through Sprite 0 collision detection	13
Interrupts	14
Hardware Register Summary	15
Sampled Sound	20
Sampled sound causes timing problems	21
"Mystery" IRQ	22
Monochrome display gets tint	22
"Zapper" Light Gun	22
"Power Pad" exercise controller	23
Video stability with dithered images	23

NINTENDO PROGRAMMER'S REFERENCE GUIDE

1

Nintendo Programming Notes Electronic Arts CONFIDENTIAL

July 14, 1989 Edition

INTRODUCTION

This document describes programming of the NINTENDO ENTERTAINMENT SYSTEM (NES) for developers. Thorough knowledge of the 6502, graphics, and sound programming are prerequisites to reading this, of course.

Instead of providing actual code, all concepts have been abstracted. You must write your own code based on the summaries provided within.

GENERAL DESCRIPTION

The NES is a dual bus architecture-- one bus controls video memory, and the other bus is used by the CPU. The CPU is a 6502 compatible chip, with on-chip support for game controller I/O, sound, and sound DMA. It runs at 1.79 MHz, with no wait states except for sampled sound DMA (optional.)

The video controller is referred to as the "PPU", or Picture Processing unit. The CPU commands the PPU through eight addresses within its memory space. Through these registers, the CPU can read or write PPU memory indirectly, set up colors and scrolling, and set up all of the other essential video parameters.

The PPU generates a 32 wide x 30 high character display. Each character is 8 x 8 pixels, of up to 4 colors per pixel. Groups of 4 characters may select one of four character color palettes.

Each character is specified by a character number between 0 and 255, with no attribute bits incorporated into the character number. This gives a full set of 256 unique characters available for use in the display.

The PPU also displays sprites. Up to 64 sprites may be displayed, but only 8 may be displayed on any given horizontal line. The sprites have their own shape set apart from the character set. Because the PPU accesses its own memory to display characters and sprites, there is no overhead to the CPU.

PPU memory can only be accessed by the CPU (through the PPU registers) during vertical blank time, or if the screen is specifically blanked. This means that code must be carefully written to take advantage of the limited vertical blank time to rewrite characters or sprites.

The NES motherboard contains no ROM. Only 2K bytes of general purpose RAM is placed on the motherboard for use by the CPU. Another 2K bytes of RAM is connected to the PPU, for two pages of video display. These two pages can be oriented vertically or horizontally relative to each other, by cartridge hardware selection. A horizontal orientation, for example, is ideal for a horizontal scrolling game.

Additional RAM or ROM must be placed in the cartridge to supply character sets to the PPU for character and sprite images. More RAM can be added to provide two extra video display pages to facilitate diagonal scrolling games.

CPU Memory Map

\$0000-\$07FF:	2K bytes RAM, located in the Nintendo base unit.
\$0800-\$1FFF:	unavailable: duplicate image of RAM, or unused
\$2000-\$2007:	PPU Registers: see hardware register summary
\$2008-\$3FFF:	unavailable: duplicate of PPU registers, or unused
\$4000-\$4017:	Sound, game controller, and IRQ control registers
\$4018-\$5FFF:	unavailable
\$6000-\$7FFF:	up to 8K bytes optional RAM expansion on cartridge
\$8000-\$BFFF:	16K bytes of bank-switched ROM on cartridge
\$C000-\$FFFF:	16K of fixed ROM on cartridge (never bank-switched)

PPU Memory Map

\$0000-\$0FFF:	character set for sprites or background characters
\$1000-\$1FFF:	alternate character set, for sprites or background
\$2000-\$23BF:	Video display page 1
\$23C0-\$23FF:	Color map selection for display page 1
\$2400-\$27BF:	Video display page 2 (if horizontal jumper set or extra RAM provided on cartridge)
\$27C0-\$27FF:	Color map selection for display page 2
\$2800-\$2BBF:	Video display page 2 (if vertical jumper is set or extra RAM provided on cartridge)
\$2BC0-\$2BFF:	Color map selection for display page 2
\$2C00-\$2FFF:	Extra video page and color map RAM (if provided on cartridge)
\$3F00-\$3F1F:	Color palette selections for characters and sprites

Screen Organization

Display Page Organization

A Nintendo display page is composed of 30 rows of 32 characters, organized as linear memory, one byte per character. The display pages are contained within PPU memory, therefore can only be accessed through the PPU

registers during vertical blank time or when the screen is blanked by software. See the PPU Memory Map for the addresses of the display pages.

Two display pages' worth of memory is provided in the NES motherboard. Using horizontal and/or vertical scrolling, the PPU will obtain part of the picture from one display page, and part from another. Additional RAM may be added for two more display pages.

Overscan

The top and bottom row, and the left-most and rightmost character or two may not be visible on some televisions or video monitors. Be sure to check your product out with several displays to ensure that you aren't placing essential information outside of the minimum visible area.

Vertical Scrolling

If the "vertical" jumper is set, the two display pages will be stacked vertically; writing higher values to the vertical scroll register causes more of display page 2 to be seen as it is scrolled into view. Changing the horizontal scroll register causes the display to wrap around on itself from the left or right edges.

To implement a vertical scrolling game, increase the vertical scroll register as the game scrolls downward. Write a new row of characters in the display page at the bottom edge of the view whenever the vertical scroll increases by another eight pixels. To scroll upward, decrease the vertical scroll register, and rewrite rows of characters on the display page at the top edge of the view.

Version 1 of the cartridge emulator hardware causes part of the character sets to be lost when using the vertical jumper. Version 2 should correct this.

Horizontal Scrolling

If the "horizontal" jumper is set, the two display pages will be organized as two side-to-side pages; writing higher values to the horizontal scroll register causes more of display page 2 to scroll into view from the right-hand edge of the display. Changing the vertical scroll register causes the display to wrap around on itself from the top or bottom.

To implement a horizontal scrolling game, increase the horizontal scroll register as the game scrolls right. Write a new column of characters in the display page at the right edge of the view whenever the horizontal scroll increases by another eight pixels. To scroll left, decrease the horizontal scroll

register, and rewrite columns of characters on the display page at the left edge of the view.

Diagonal Scrolling

A diagonal scrolling effect can easily be implemented with the jumper set for horizontal scrolling. Rewrite characters in the leading column of the view as the horizontal scroll is changed. Rewrite characters in the wraparound row (at the top or bottom edge of the view) as the vertical scroll is changed. Some software-driven blanking may be needed to hide part of the wraparound area. To implement this, use the split-screen technique discussed later.

Expanded Display RAM

If extra RAM is provided on the cartridge, four display pages will be available, organized as a large square. The horizontal and vertical scroll registers can be used to pan around the four screens. When the scrolling extends beyond an edge of the four screens, the display is wrapped around from the other edge.

Video Timing

The screen is composed of 262 horizontal lines, of which 240 are visible, and 22 are retrace time. The beginning of this retrace time is when the vertical blank NMI occurs. The CPU can initiate data transfers to PPU memory at any time during these 22 lines without disrupting the display.

Each horizontal line takes $113 \frac{2}{3}$ clock cycles of the 6502 to display. This yields a vertical retrace time available of $(22 * 113.66...) = \text{exactly } 2500$ cycles. During the active display (240 lines), about 25 of the 113.66... cycles are horizontal retrace time, and are invisible. Because the DMA address register is also used as the video counter, the CPU can't access PPU memory during horizontal retrace without resetting the video counter with a new base address afterward. Even if you write a new base address, there's not enough time during horizontal blank time to do any DMA without damaging the next line of the display. The best solution for writing data to PPU memory during display time is to leave an entire line of the display blank, and write data during the blanked time.

Character Data Organization

Background characters consist of an 8 x 8 pixel array. Each pixel may be one of four colors, requiring 2 bits per pixel. Therefore a total of 128 bits, or 16 bytes, are required to define a character. Characters are defined in a contiguous block of 16 bytes. The definition is structured similar to IBM EGA graphics. The character has two bit planes. The first eight bytes define the first bit plane and the second eight bytes define the second bit plane. Consider the following character, with each number defining a pixel's two bit color value (0 - 3):

```

0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 2 0 0 0 0 0
0 0 0 3 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 2 0 0
0 0 0 0 0 0 3 0
0 1 2 3 0 1 2 3

```

A data statement describing this character would look like the following (using binary constants):

```

; first plane
db  %00000000
db  %01000000
db  %00000000
db  %00010000
db  %00001000
db  %00000000
db  %00000010
db  %01010101

; second plane
db  %00000000
db  %00000000
db  %00100000
db  %00010000
db  %00000000
db  %00000100
db  %00000010
db  %00110011

```

The background character set consists of 256 characters requiring a total of 4K bytes of memory. This memory is on the cartridge, and may be either RAM or ROM. The background character set may be mapped starting at \$0000 or \$1000. Mapping is controlled by PCR1, bit 4. When bit 4=0, the character set is mapped to \$0000-\$0FFF, otherwise the set is mapped to \$1000-\$1FFF.

Even though character mapping is restricted to these two address ranges, several cartridges offer bank switching mechanisms for the background character set memory. This allows access to several different background character sets.

A game could, for example, switch character sets via bank switching part-way down a title screen to allow more than 256 characters to be used for the image.

Screen Colors

The NES has four color sets made up of four colors each for use by the background characters. Each color is described by a byte where

Bits 3..0 - primary color
Bits 5..4 - intensity of color
Bits 7..6 - not used

The primary colors are:

0	Gray
1	Blue 1
2	Blue 2
3	Blue 3
4	Purple
5	Red/Pink
6	Red
7	Red/Orange
8	Brown/Yellow
9	Green 1
A	Green 2
B	Green/Turquoise
C	Cyan
D	Gray 2
E	Black
F	Black

The color sets are mapped as follows (addresses are in PPU memory):

Color set registers:

\$3F00: unused
\$3F01: color set #1 color for bit combo "01"
\$3F02: color set #1 color for bit combo "10"
\$3F03: color set #1 color for bit combo "11"
\$3F04: unused
\$3F05: color set #2 color for bit combo "01"
\$3F06: color set #2 color for bit combo "10"
\$3F07: color set #2 color for bit combo "11"
\$3F08: unused
\$3F09: color set #3 color for bit combo "01"
\$3F0A: color set #3 color for bit combo "10"
\$3F0B: color set #3 color for bit combo "11"
\$3F0C: unused
\$3F0D: color set #4 color for bit combo "01"
\$3F0E: color set #4 color for bit combo "10"
\$3F0F: color set #4 color for bit combo "11"

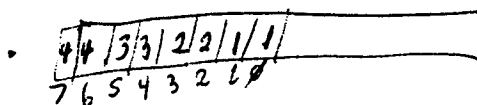
Border/Background color:

\$3F10: color for all sets for bit combo "00" and border/blanked

Color sets are assigned to characters as follows: First divide the screen into blocks of 4x4 characters starting at the top left of the screen. You will end up with 64 blocks with one color memory byte

assigned to each block. Further divide each block into 4 subblocks of 2x2 characters. Number these 1 through 4 starting at the top left subblock. The first block on the screen would look as follows (character positions are in decimal):

	1	2	
	0		
	00	01	02 03
	32	33	34 35
3	64	65	66 67
	96	97	98 99
		4	



Subblock 1 consists of characters 00, 01, 32, and 33.
 Subblock 2 consists of characters 02, 03, 34, and 35.
 Subblock 3 consists of characters 64, 65, 96, and 97.
 Subblock 4 consists of characters 66, 67, 98, and 99.

Since there are four color sets, two bits are required to select a color set for a subblock. Four subblocks require a byte and therefore each block is assigned a byte from color memory. The bit pairs are arranged as follows:

Subblock 1: bits 1,0
 Subblock 2: bits 3,2
 Subblock 3: bits 5,4
 Subblock 4: bits 7,6

It follows that each subblock is restricted to three colors plus background. The respective color memory bit pairs selects one of the four color sets; the pixel bit pairs within the characters select the color from the chosen color set.

A color memory value of \$D1 (%11010001) would assign color set 2 to subblock 1; color set 1 to subblock 2; color set 2 to subblock 3; color set 4 to subblock 4.

These color restrictions are quite severe, especially in light of the incompatibility with existing paint programs. The four colors per subblock restriction disallows straight conversions from existing multi color pictures (such as a Deluxe Paint EGA picture.) Pictures should be drawn with these restrictions in mind.

Sprite Programming

The NES has 64 physical sprites. Each sprite consists of one or two 8 x 8 pixel characters (depending on the sprite mode.) Characters are defined exactly in the same way as background characters. The sprite character set may be mapped to either \$0000-\$0FFF (PCR1 bit 3 = 0), or \$1000-\$1FFF (PCR1 bit 3=1).

Each sprite is defined by 4 bytes as follows:

Byte 0: Sprite Y position
Byte 1: Sprite character number
Byte 2: Sprite color, priority, and orientation.
 Bits 0 and 1 are used to select a color set. Bit 5, when set, gives background characters priority over sprites. Bit 7, when set, flips the sprite horizontally. Bit 6, when set, flips the sprite vertically. Bits 7 and 6 may be used together for diagonal mirroring.
Byte 3: Sprite X position

Sprite Parameter Transfer

Sprite data is generated by the CPU and placed in a page of RAM (any page of available RAM is acceptable.) One page is required, because there are 64 sprites, with 4 bytes of parameter data each. During vertical blank a DMA command is issued by the CPU to transfer the entire sprite data page to the PPU. This command must be issued while blanking is turned on.

A simple way to initiate this transfer would be to first write a \$00 to PPCR (\$2003), then the page number of source RAM to DPNR (\$4014). The write to DPNR initiates the DMA transfer, which requires 512 clock cycles to complete.

Writing a \$00 to PPCR defines the initial sprite number to be sprite 0 to receive the DMA data. Writing another number (always a multiple of 4) will change the priority of the sprite table. For example, writing a \$04 to PPCR then writing \$0x to DPNR will cause data from \$x00-\$x03 to transfer to sprite 1 instead of sprite 0, lowering its priority. \$x04-\$x07 will transfer to sprite 2, and so on. This has the effect of lowering the priority for all sprites except the last at \$xFC-\$xFF, which will now have the highest priority.

Sprite limitations - multiplexing

The NES limits sprite usage to eight sprites per horizontal line. Any more sprites per line get ignored by the PPU and do not show up on the screen. This limitation may be overcome by using sprite multiplexing. It is a commonly used technique in NES games, easily recognized by the annoying flicker it produces. However, it is the only way to show more than eight sprites per line. Many scrolling games use sprites to provide a stationary score/status display. Because this may use up to eight sprites, sprite multiplexing becomes inevitable. If sprite multiplexing is not desired, the game should be carefully designed with the eight sprite limitation in mind at all times.

Sprite Priority

Sprites are prioritized amongst themselves by their position index. The lower the index, the higher the priority. This applies to the eight sprite limit: higher priority sprites are always seen. If you try to display sprites 1 through 10 on the same line, sprites 9 and 10 will be blanked. There is no hardware mechanism available for detecting this condition.

By writing a different initial sprite # to PDPC, the sprite parameter number, the priority of the data being copied from RAM to the sprite hardware will be different. For example, if a #\$80 is written to PDPC instead of the #0 in the example, the sprite whose parameter data is at \$x80 will be sprite 0 (highest priority.) Sprite multiplexing can be easily accomplished by changing the value written to PDPC every other vertical blank, and by organizing the sprite data so that sprites which may be on the same lines are always in opposite halves of the sprite RAM page.

Disabling Sprites

One way to disable sprites is to set the Y coordinate to a value of \$F0 through \$F8 (only \$F0 will work in double height mode.) This will always place the sprite off the visible screen, because the visible screen is only in a range of Y coordinates \$00-\$EF anyway.

A sprite may also be disabled by assigning it a character which is completely transparent (all zeros.)

Background Priority

Sprite to background priority is controlled by bit 5 of the sprite color/orientation/priority parameter for each sprite. If this bit is 0,

sprites appear in front of the background (most sprites will be like this.) If this bit is set, any nonzero background data will appear in front of the sprite.

Sprite Color Selection

Each sprite may choose from one of four color sets reserved for sprites. Any sprite may choose any of the four color sets, so sprite overlays can be used to enhance the number of colors an image may have.

Sprite Overlays

To achieve more than the three colors plus transparency in a sprite, one can assign different color sets to two or more sprites, then place them in the same screen location. Each sprite can contain part of the desired multi-color image. However, this technique more rapidly reaches the eight sprite per horizontal line limitation of the PPU.

Sprite 0 Collision Detection

A collision between sprite 0 (the highest priority sprite) and the background can be detected. The collision is detected when a non-zero pixel in sprite 0 occupies the same screen position as a non-zero background character's pixel. At the moment of this collision, bit 6 of PSTR (\$2002) will be set. If a collision is detected, this bit will remain set until the end of vertical blank, when it will be cleared automatically.

The collision will be detected even if the background has priority over sprite 0.

Sprite Priority vs. Background Priority

A sprite which gives priority to the background still retains its priority relative to other sprites. If the sprite is obscured by the background, then lower priority sprites in the same position will also be obscured.

This feature is very useful to implement a general purpose background transparency scheme. Simply use a high priority sprite with the same shape as a background object (a chair, for example.) Give the background priority over this sprite. Lower priority sprites which move across the chair will appear "behind" the chair, because they will be masked by the higher priority sprite.

Problem with blanking and sprites

If the blanking bits are turned on during active screen area (to rewrite a color palette register, for example,) then sprite DMA may not occur for sprites beginning on lines being blanked. The result is blanked or flickering sprites.

Sprite Recycling

Sprites can be re-used. During a split screen operation, for example, turn blanking on. Do the sprite data transfer as usual, first writing a value to location \$2003, then \$4014. This will take a little less than 5 scan lines (512 clock cycles.) The DMA is transparent to the CPU, so blanking can be turned off immediately after initiating the transfer.

Alternatively, to update a single sprite, write its address (sprite # * 4) to \$2003, then data to \$2004 for as many bytes as are needed. Entire sprites are updated by writing multiples of 4 bytes of data. Writing less than 4 bytes leads to hardware glitches. As many sprites as you wish can be updated this way, but it may still be more efficient to use the DMA transfer feature mentioned above. The blanking bits must be set during this transfer.

Sprite 0 may be recycled as a normal sprite, but the collision detect bit will remain set until the end of vertical blank time anyway. Therefore, recycling sprite 0 will *not* allow detection of more sprite collisions.

Sprite Double-Height Mode

All sprites may be either a single character (8 x 8) or two characters tall (16 x 8). When the sprites are set to this double-height mode, the character number used to define the sprite shape is modified. The least significant bit of the character number is ignored. The sprite will then use a pair of characters starting at the character number specified. The first character appears on top, and the second character appears immediately below the first.

If the horizontal flip bit is set for a sprite, both characters are flipped horizontally. If the vertical flip bit is set, not only are both characters flipped vertically, but the second character now appears on top. In other words, both characters are grouped as a single shape and the whole shape is flipped when the parameters specify.

Sprite Colors

Sprite color set memory is organized the same as character color memory. Remember that bits 0 and 1 of each sprite's color/priority/orientation byte select which color set the sprite will use. Following is a map of the color locations in PPU memory:

```
$3F11: color set #1 for bit combo 01
$3F12: color set #1 for bit combo 10
$3F13: color set #1 for bit combo 11
$3F14: unused (bit combo 00 is always transparent)
$3F15: color set #2 for bit combo 01
$3F16: color set #2 for bit combo 10
$3F17: color set #2 for bit combo 11
$3F18: unused
$3F19: color set #3 for bit combo 01
$3F1A: color set #3 for bit combo 10
$3F1B: color set #3 for bit combo 11
$3F1C: unused
$3F1D: color set #4 for bit combo 01
$3F1E: color set #4 for bit combo 10
$3F1F: color set #4 for bit combo 11
```

As in the case of all PPU memory, blanking must be turned on before writing to these addresses through the PMAR and PMDR registers.

Split Screen through Sprite 0 collision detection

To use this feature, Sprite 0 should be placed at the spot on the screen where split screen operation is desired. Sprite 0 should be at the right-hand edge of the screen, at X coordinate \$F8-\$FF. The screen should be organized so that a solid non-zero character image appears at the same position as sprite 0. By polling bit 6 of PSTR (\$2002), the exact moment of sprite collision can be detected. The scroll registers can be reset at this point.

Bit 6 of PSTR remains set until the END of vertical blank time. This is useful should you want to synchronize your code to the beginning of the active screen, and perform multiple changes on the video throughout the screen.

Bit 6 may conveniently be examined with an AND, ASL, or BIT instruction.

Only the horizontal scroll register affects the scrolling during screen display time. To reset the vertical scroll position, *you must store a new base address* to display data from to the DMA Address register (PMAR, location \$2006;

store high byte first.) The first scan line of the new data may be lost or garbled, depending on how long the delay is between detecting a sprite collision and rewriting the scroll registers.

The collision detect bit remains set until the next vertical blank, hence only one collision can be detected. To split the screen more than once, carefully calculated precise delays after the initial split are needed (113 and 2/3 CPU clock cycles per scan line.)

Interrupts

There are two potential sources of interrupts in the NES system: an NMI interrupt which can occur at the beginning of vertical blank, and an IRQ interrupt, which can be programmed to happen at the end of the performance of a sampled sound.

To enable the vertical blank NMI, write a value to PCR1 (\$2000) with bit 7 set.

When responding to a vertical blank NMI, read PSTR (\$2002). This clears the NMI signal, and allows further interrupts to occur.

To enable the sampled sound end-of-sound IRQ, enable IRQ's in the CPU with a CLI instruction, then write a value to V5R1 (\$4010) with bit 7 set. The IRQ will be generated at the end of the playback of any sampled sound.

When responding to the IRQ, write a value between \$40 and \$FF to location \$4017. This clears the source of the IRQ signal, thereby preventing the IRQ from re-interrupting the CPU when the CPU exits the IRQ handler.

Hardware Register Summary

\$2000 (PCR1): PPU CONTROL REGISTER 1

bit 0 : MSB of horizontal scroll offset

If set to a 1, the horizontal scroll will be extended 256 pixels to the right. (scrolls to the right-hand screen if horizontal jumper set)

bit 1 : MSB of vertical scroll offset

If set to a 1, the vertical scroll will be extended 240 pixels downward (scrolls to the bottom screen if vertical jumper set or extra PPU RAM provides for 4 screens)

bit 2 : Auto-increment mode for PPU memory access

If set to a 0, accesses to PPU memory will happen linearly (the address will advance by 1 each read or write.) If set to a 1, the address will advance by 32 each time, useful for drawing columns of characters.

bit 3 : Sprite Character Memory Address

Selects whether sprite character data should be read from PPU memory at \$0000-\$0FFF (bit 3=0) or \$1000-\$1FFF (bit 3=1)

bit 4 : Background Character Memory Address

Selects whether background character data should be read from PPU memory at \$0000-\$0FFF (bit 4=0) or \$1000-\$1FFF (bit 4=1)

bit 5 : Sixteen pixel tall sprite mode

If set to 1, sprites will be 16 pixels high instead of 8. The least significant bit of the character number for each sprite is ignored, and characters are used in even-odd pairs to provide the 16 lines of data that the sprite will use. The flip and mirror bits work properly in this mode. NO sprite entries are sacrificed-- all 64 sprites are available for use.

bit 6 : unused/undocumented (set to 0)

bit 7 : Vertical blank NMI interrupt enable

Set this bit to 1 if you want an NMI interrupt to occur at the beginning of vertical blank time. To reset the source of the NMI, read \$2002 (PSTR).

\$2001 (PCR2): PPU CONTROL REGISTER 2

bit 0 : Monochrome mode

This bit, if set to 1, disables color in all palettes. This mode isn't very useful, because you can easily rewrite the color set memory directly and get the same effect.

bit 1 : Border width extend bit

If set to 0, the left border of the screen is extended to the right by 8 pixels (1 character column.) If the game has some tearing or garbage at the edge because of scrolling problems, this bit can be reset to hide it.

bit 2 : unused/undocumented (set to 1) →

*Border bit for
Sprite*

bit 3 : Background enable bit

Set this bit to 1 to enable background character graphics, or set to 0 to disable them. This bit MUST be set to 0 when reading or writing PPU memory.

bit 4 : Sprite enable bit

Set this bit to 1 to enable sprites, or set to 0 to disable them. This bit MUST be set to 0 when reading or writing PPU memory, or updating sprite parameters.

bits 5-7: Color and intensity adjustments to the palette
These bits affect the entire palette, for sprites and for background. Bit 5, if set, tints the palette darker red. Bit 6 tints the palette darker green. Bit 7 tints the palette darker blue. If all three bits are set, the palette colors remain their default hues, but are darkened by 50%.

During split screen operation, any of these parameters can be changed, and they will take effect immediately. This may be useful, for example, if 8 pixel tall sprites are desired for a status window, but 16 pixel tall sprites are needed for game characters. Palette tint adjustments can also be made, to add more or different colors to a status window.

\$2002 (PSTR): PPU STATUS REGISTER (read only)

bits 0-5: undocumented/unused

bit 6: collision detection bit

This bit detects collisions between sprite 0 and the background. Sprite 0 still detects collisions if its background priority bit is set (background has priority over the sprite.) The background must be non-zero to allow the sprite to detect the collision. The bit will remain set until the END of vertical blank time.

bit 7 : Vertical blank occurred bit

This bit will be set when vertical blank starts. Reading this register clears bit 7. This bit must be cleared by such a read in order to re-enable NMI interrupts for the next vertical blank.

When this register is read, the high/low flag for PMAR (\$2006) is reset (the next write to PMAR will definitely be the high order address.) The horizontal/vertical flag for the PSOR scroll register (\$2005) is also cleared, ensuring that the next write to PSOR will be the horizontal scroll amount.

\$2003 (PPCR): SPRITE REGISTER DESTINATION ADDRESS

This is the single byte address within the sprite hardware to start writing to when storing from \$2004. Normally set to 0 when doing a DMA transfer by writing the page # to DPNR (\$4014). To switch the priority of the first and second 32 sprites, though, a \$80 could be written here. To manually rewrite the vertical position for sprite 1, (for example,) store a \$04 here, then the new vertical position to \$2004. It's recommended that if you update one parameter for a sprite, you update all four. Note that the DMA will not cross a page boundary. The low order address may change, but the DMA will always occur on the page specified by DPNR.

\$2004 (untitled): SPRITE PARAMETER DATA PORT (bidirectional)

The DMA feature activated by DPNR writes 256 bytes of data here. Each write to \$2004 writes data to a sprite hardware register specified by PPCR (\$2003), then increments the PPCR counter by 1. Each read will read data from the sprite hardware register.

\$2005 (PSOR): PPU SCROLLING REGISTER (horizontal, vertical)

This register is used to choose how much to scroll the screen horizontally and/or vertically. The first write to this register

specifies the horizontal displacement; the second write specifies the vertical. Vertical scroll amounts greater than 240 (\$F0) are invalid. The next logical scroll amount after \$EF is \$100 (use the vertical scroll MSB in PCR1, bit 1.)

During active video area, the horizontal scroll amount may be rewritten, but the vertical scroll amount is ignored by the PPU. To get a change in the vertical scroll during display time, rewrite the PPW display address through PMAR (\$2006) instead.

\$2006 (PMAR) : PPU DMA ADDRESS REGISTER/DISPLAY ADDRESS

DMA Address register (high, low)

In order for the CPU to access PPU memory, it must write the desired address here, high order byte first. Data then may be read or written to PMDR (\$2007). The first byte of a read sequence should be discarded.

Blanking **MUST** be set (PCR2, \$2001, bits 3 and 4 set to 0) before writing values to PMAR or PMDR.

Video display address (high, low) In addition to its use as a DMA address register, PMAR contains the active video counter used to fetch display data. If PMAR is written during active video time, even while the display is blanked, it'll need to be rewritten with a new address to display characters from after the program is through using PMAR for DMA. The new address will be *unaffected by the setting of the vertical scroll register*, unlike at vertical blank time.

Before the end of vertical blank, write a pair of \$00's here to reset the video address offset to \$0000 before video starts, otherwise the video will start from the wrong place.

\$2007 (PMDR) : PPU DMA DATA REGISTER (bidirectional)

After an address is written to PMAR (\$2006), and the autoincrement mode set to 1 or 32 (PCR1, \$2000, bit 2), data may be written or read from here. After each write or read, the DMA Address is incremented by the autoincrement amount. When reading, the first byte is invalid and may be discarded.

\$4000 (V1R1) : Voice 1 Duty cycle, amplitude, sound length

\$4004 (V2R1) : Voice 2 Duty cycle, amplitude, sound length

The registers above are identical in format. The format is:

Bits 0-3 : Amplitude (AAAA)

A 0000 specifies minimum volume; A 1111 specifies maximum.

Bits 4-5 : Envelope mode

The combination of these two bits determines what kind, if any, of envelope will be heard:

00 - the length code (V1R4 or V2R4, bits 3-7) specifies the length of a one-time enveloped sound. AAAA specifies the initial volume.

01 - the length code specifies the length of a one time NON-enveloped sound. AAAA specifies the volume for the entire duration of the sound.

10 - the length code specifies the length of a repeating enveloped sound. AAAA specifies the initial volume.
 11 - AAAA specifies an absolute volume for the sound. This mode is most useful, because it allows the construction of complex envelopes by software control.
Bits 6-7: duty cycle of waveform
 These two bits control the ratio of high-to-low in the sound's waveform. Changing this parameter changes the tonal quality of the sound. The values are:

00	12.5%	
01	25%	
10	50%	These numbers are supplied by Arti
11	75%	and need to be verified.

\$4001 (V1R2): Voice 1 Frequency sweep controls

\$4005 (V2R2): Voice 2 Frequency sweep controls

The formats of these registers are:

ExxxDCCC, where

E (bit 7) is the sweep enable bit (set to 0 to disable sweeping.)

xxx bits (bits 4-6) are "don't care" bits

D (bit 3) is the direction, and

CCC (bits 0-2) is the 3 bit "sweep delay" or frequency cut-off value.

When the frequency sweep enable bit "E" is set to zero, the sweep delay bits ("C" bits) should be set to 111. Otherwise, the frequency cut-off feature may silence low or high pitched notes.

\$4002 (V1R3): Voice 1 Frequency lower 8 bits

\$4006 (V2R3): Voice 2 Frequency lower 8 bits

\$400A (V3R3): Voice 3 Frequency lower 8 bits

The lower 8 bits of the frequency (pitch) for the voices are specified by these registers.

\$4003 (V1R4): Voice 1 Frequency upper 3 bits/envelope length

\$4007 (V2R4): Voice 2 Frequency upper 3 bits/envelope length

\$400B (V3R4): Voice 3 Frequency upper 3 bits/envelope length

bits 0 - 2: Frequency (pitch), most significant 3 bits

bits 3 - 7: Length code. If the envelope mode specifies a length, this length code is used. The code is used to index a lookup table of actual durations. This table bears a stronger resemblance to random numbers than useful durations.

\$4010-\$4013: Sampled Sound parameters (see section on sampled sound)

\$4014 (DPNR): SPRITE DMA START ADDRESS (high)

This is the high order address (most significant byte) where sprite parameters will be read from. Writing a value here initiates a 256 byte transfer from the specified page to the sprite parameter data port (\$2004).

\$4015 (VMER) : SOUND MASTER ENABLE REGISTER

This is the master control register for turning voices on and off.

bit 0 : voice 1 enable if bit set to one; disable if set to zero

bit 1 : voice 2 enable

bit 2 : voice 3 enable

bit 3 : voice 4 enable

bit 4 : Sampled sound voice enable

To retrigger an envelope, or restart a sampled sound, first clear then set the enable bit for that voice.

\$4017 (Game controller port 2):

Must be written with a value between \$40 and \$FF during system initialization, otherwise spurious IRQ interrupts will be generated.

Sampled Sound

Sampled sounds are composed of single bit samples. These samples are just clipped raw audio data, no better than the PC or Apple II sampled sounds. 8 samples are packed into each byte, with the least significant bit "played" first. Data is retrieved sequentially from memory starting at the base address specified in V5R3, for the length in V5R4. The playback rate and loop modes are set in V5R1. V5R2 is the shift register through which the hardware shifts bits in the byte currently being played.

Data can only be played from the ROM area \$C000-\$FFFF. This is the unbanked portion of standard cartridge ROMs, unfortunately, limiting the total samples available for play to a bit less than 16k bytes.

Sampled sound hardware registers:

\$4010 (V5R1): Sampled sound Playback rate, loop mode, IRQ enable
bits 0 - 3: DMA playback rate. This 4 bit code tells the sound chip how fast to play the sampled sound. Codes from 0..7 are a C-major scale for music. Higher numbers are used for higher playback rates. The sample rates are:

1111 (F)	33153 Hz	0111 (7)	8364 Hz
1110 (E)	24860 Hz	0110 (6)	7920 Hz
1101 (D)	21310 Hz	0101 (5)	7047 Hz
1100 (C)	16888 Hz	0100 (4)	6258 Hz
1011 (B)	13983 Hz	0011 (3)	5593 Hz
1010 (A)	12606 Hz	0010 (2)	5264 Hz
1001 (9)	11187 Hz	0001 (1)	4710 Hz
1000 (8)	9420 Hz	0000 (0)	4182 Hz

(determined by measuring time between samples read from memory on the logic analyzer)

bit 6: loop mode

If set to a 1, the sound will repeat itself continuously while the master enable bit at \$4015 is set. If set to a zero, then the master enable must be turned off then on again to initiate a sound.

The playback rate can be adjusted any time during playback of a sample and will take effect immediately.

bit 7: IRQ enable

If set to 1, an IRQ interrupt will be generated at the end of a played sampled sound. This bit MUST be set to 0 at some time before the CPU returns from the interrupt. If the loop bit is set, the sound will continue looping, and generates an IRQ at the end of each repetition of the sound.

\$4011 (V5R2): Sampled sound shift register

This is the 8 bit sample register which is sent to the audio hardware, 1 bit at a time. The sampled sound DMA circuitry reads the samples from ROM, then writes them here. Samples aren't played from this register when DMA is disabled, so there isn't any practical need for the user to write here.

\$4012 (V5R3): Base address of sampled sound

The formula for the base address is: $\$C000 + (V5R3 * \$40)$. This means that samples must start on a boundary of \$40, and can only reside in the area from \$C000-\$FFC0.

\$4013 (V5R4): Length of sample - 1.

The length seems to be in multiples of \$10 bytes. The formula for the sample length is: $(V5R4 + 1) * \$10$ = sample length in bytes. This limits the sample length to \$1000 bytes.

\$4015 (VMER): VOICE MASTER ENABLE REGISTER

Bit 4 : Sampled sound master enable bit. Set this bit first to "0", then to "1" to play the sound whose parameters have been set up in V5R1-V5R4.

Sampled sound causes timing problems

The DMA from sampled sound hardware will steal 2 bus cycles from the CPU out of every 400-3400, depending on the playback rate. This will foul up fixed timing loops used for display modification. The solution is either to avoid sampled sound in these situations, or compensate for them in your timing.

IRQ interrupts generated at the ends of sampled sounds can affect timing even more.

Sampled sound DMA does NOT occur when the voice is disabled (bit 4 of \$4015 is zero.)

"Mystery" IRQ

If location \$4017 is written with any value less than \$40, IRQ interrupts will occur every 1/60th of a second. However, the interrupt gradually loses sync with the video hardware if location \$2002 is polled for split screen operations.

The purpose of this interrupt is unknown, but it may be an external hardware feature that was partially unimplemented and consequently generates IRQ from noise on the circuit board. It may not be implemented on all Nintendo systems, either.

Monochrome display gets tint

If the monochrome bit (\$2001, bit zero) is set, the individual colors lose their RGB content, but the palette tint adjustment bits (\$2001, bits 5-7) still tint the picture.

"Zapper" Light Gun

The light gun should be plugged into port 2. There are only 2 bits of data provided by the gun: the fire button and the optical sensor. The optical sensor is focused on the television. As the light from the TV phosphor fades away, the sensor reads a "1". When the electron gun strikes the phosphor again, the sensor reads a "0". By timing, you can know when (or if) the gun was focused on the desired target.

\$4017 JOY2PRT:

Bit 3 : Optical sensor. Low (0) when light is being detected. Blank the whole screen except for your targets, and see if the gun collided. Or, synchronize your timing to the NMI, and determine the physical screen position which the sensor picks up. The stability is improved if you average 2 or 3 samples in a row, and use the maximum brightness for your targets. I'm not sure how reliable the measured approach is versus blanking the screen. The stability isn't very good with the timed measurement code I've tested. It could be caused by ambient, and unfocused CRT light. Further tests should be made when a light gun title is proposed. In the worst case, the blanked screen approach works reliably enough to use. Be sure to test a light gun code on many televisions, because phosphor persistence varies widely and has a significant effect on the gun circuitry.

Bit 4 : Fire trigger. High (1) when the gun is fired. In normal use, firing the gun will pulse the trigger switch for about 1/20 of a second. However, the trigger can be manipulated by a skilled player so that the button is always in the on, or firing position. Allow for this in your code.

"Power Pad" exercise controller

The power pad is a matrix of on/off switches. The switches are passed through a pair of shift registers, and read in a manner similar to the joystick. The bit assignments are as follows (joystick port 2 is assumed, because games require the pad to be plugged in there.)

\$4017 JOY2PRT:

Bit 3: serial input data. When JOY2PRT is read, a new bit of data from the pad is shifted in. A "0" bit means the pad is not being pressed. A "1" means someone is standing on it.

Power Pad Bit 3 Data:

First bit read	Blue #2
Second bit read	Blue #1
Third bit read	Blue #5
Fourth bit read	Blue #9
Fifth bit read	Blue #6
Sixth bit read	Blue #10
Seventh bit read	Red #11
Eighth bit read	Red #7

\$4017 JOY2PRT:

Bit 4: Serial input data. Read data from this bit using the SAME bytes read while processing bit 3. In other words, only read JOY2PRT a total of 8 times.

Power Pad Bit 4 Data:

Bit	Hex	Description
First bit read	Red #4	
Second bit read	Red #3	
Third bit read	Red #12	
Fourth bit read	Red #8	
Fifth bit read	unused	
Sixth bit read	unused	
Seventh bit read	unused	
Eighth bit read	unused	

Video stability with dithered images

Fringing and flickering effects caused by dithering seem to stabilize or go away if the screen is scrolling by 2 pixels per display frame instead of 1. In other words, if fringing is a problem, always scroll on an even pixel boundary.

; ACTUAL SOUND ROUTINES THAT ARE CALLED FROM ABOVE STRUCTURE

; NES SOUND STRUCTURE DEFINITION:

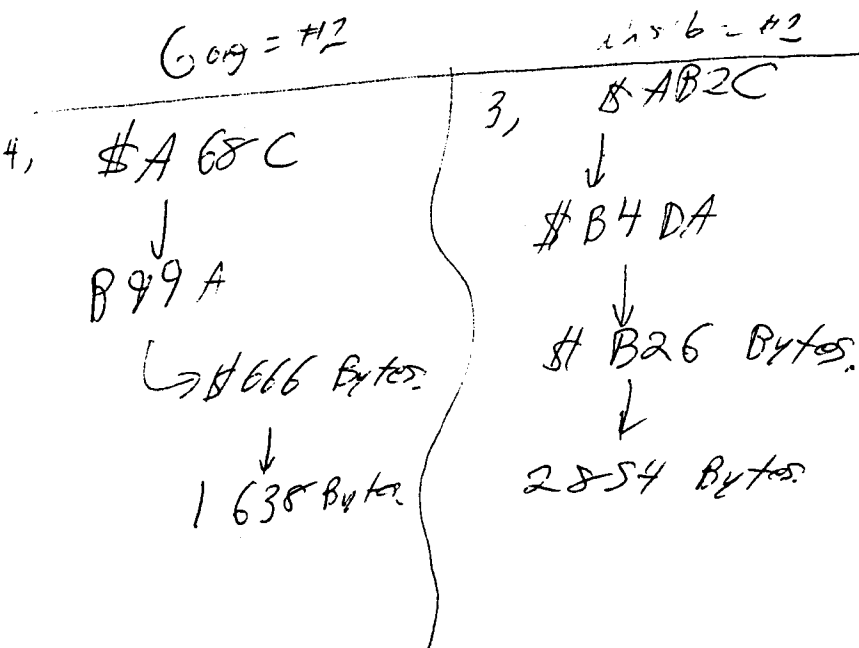
;WRA0 = %DDTEAAAA
D = Duty Cycle
T = Timer Mode (always be 0), timer on
; E = Envelope, 0 = Fading sound - 1 = Steady Sound
; A = Amplitude/Volume (or envelope step speed)
;
;WRA1 = ESSSDBBB
; E = Sweep Enable
; S = Sweep Speed (speed - 1), how many quarter-NMI's between each pitch c
; D = Add/Subtract Direction/Mode, 0 = Pitch Increases - 1 = Pitch Decreases
; * In subtract mode, below "B" should only be 1 or 2
; B = Bits Frequency Is Shifted By Per Update
;
;WRA2 = Frequency of Notes (look up from FreqTabl.New)...small # = Higher Pitch
;
;WRA3 = LLLLLFFF
; L = Length of Sound, how many frames (NMI's) the sound will last...look
; F = Last High Bits of Sound Frequency - Generally, don't worry about t
; *NOTE: Writing to this byte begins the sound that has just been defined

explosion_sound

```
lda #%00001111
sta VMER                      ;Clear Sound Master Register

lda    #%11001111            ;slow envelope, single shot, 12.5% duty cycle
sta    WRA0                  ;voice B (square wave)
lda    #%10010010            ;sweep
sta    WRA1
lda    #150                  ;start at this high pitch
sta    WRA2
lda    #00100000              ;length=126
sta    WRA3                  ;start sound

rts
```



```

;TSYNC EQU $C001 ;Bit 0: (Load and Synchronize Timer)
; ; 0=Timer counts horizontal display lines
; ; 1=Timer counts CPU clock (PHI2) divided by 4
; ; A write to this address forces the value in
; ; TLAT ($C000) to be written to the interval
; ; timer (resets interval timer as result)
DITIRQ EQU $E000 ;A Write to this address Disables the IRQ Timer
EITIRQ EQU $E001 ;A Write to this address Enables the IRQ Timer
;NOTE: Below Register Select Equates are commented out because they are
; identical to the below MMC-3 Register Select Equates
;Register Select Equates (Assumes Char. Map Move is set to 0)
; Bank #0 (2Kbyte bank/128 Char. Bank) -suggested as Background Bank
; CBSR0 EQU $00 ;Character Bank Select Register 0
; CBSR1 EQU $01 ;Character Bank Select Register 1
; Bank #1 (1Kbyte bank/64 Char. Bank) -suggested as Sprite Bank
; CBSR2 EQU $02 ;Character Bank Select Register 2
; CBSR3 EQU $03 ;Character Bank Select Register 3
; CBSR4 EQU $04 ;Character Bank Select Register 4
; CBSR5 EQU $05 ;Character Bank Select Register 5
; Program Bank Selection (Assumes Program Map Mode is set to 0)
; PBSR0 EQU $06 ;Program Bank Select Register 0 ($8000-$9FFF)
; PBSR1 EQU $07 ;Program Bank Select Register 1 ($A000-$BFFF)
PBSR2 EQU $0F ;Program Bank Select Register 2 ($C000-$DFFF)
;*Note: For MMC-3 Compatibility, initialize PBSR2 to $3E (%00111110) before cal
; any routine in $C000 range.
;*Note: For Mimic Compatibility, set Scroll and Disable IRQ before setting
; Register 7 (PBSR1)
;*Note: To insure FULL compatability with any ROM size, you could OR the
; Register Select byte with the unused bits set high. For the various
; size cartridges, use the below Equates
; PROGRAM Select:
UnusedBits_4MEGABIT_PRG EQU %00000000 ;512Kbyte,64 bank - 48 Pin RAMBO
edBits_2MEGABIT_PRG EQU %00100000 ;256Kbyte,32 bank - RAMBO
edBits_1MEGABIT_PRG EQU %00110000 ;128Kbyte,16 bank - RAMBO/NAMCO 109
unusedBits_512KBIT_PRG EQU %00111000 ;64Kbyte,8 bank - RAMBO/MIMIC
UnusedBits_256KBIT_PRG EQU %00111100 ;32Kbyte,4 bank - RAMBO/MIMIC
UnusedBits_128KBIT_PRG EQU %00111110 ;16Kbyte,2 bank - RAMBO/MIMIC
UnusedBits_64KBIT_PRG EQU %00111111 ;8Kbyte,1 bank - RAMBO/MIMIC
; CHARACTER Select:
UnusedBits_2MEGABIT_CHR EQU %00000000 ;256Kbyte,16K stamps - RAMBO
UnusedBits_1MEGABIT_CHR EQU %10000000 ;128Kbyte,8K stamps - RAMBO/NAMCO
UnusedBits_512KBIT_CHR EQU %11000000 ;64Kbyte,4K stamps - RAMBO
UnusedBits_256KBIT_CHR EQU %11100000 ;32Kbyte,2K stamps - RAMBO/MIMIC
UnusedBits_128KBIT_CHR EQU %11110000 ;16Kbyte,1K stamps - RAMBO/MIMIC
UnusedBits_64KBIT_CHR EQU %11111000 ;8Kbyte,512 stamps - RAMBO/MIMIC

;
;
; *** MMC-3 ***
;
;Multi-Memory Controller 3 (MMC3) Equates
; PROGRAM BANKING:
; The MMC3 splits the program area into four 8Kbyte banks
; Area 1 (8Kbyte) $8000 - $9FFF - set through PBSR0
; Area 2 (8Kbyte) $A000 - $BFFF - set through PBSR1
; Area 3 (8Kbyte) $C000 - $DFFF - fixed at second highest bank
; Area 4 (8Kbyte) $E000 - $FFFF - fixed at highest bank
; CHARACTER BANKING:
; The MMC3 splits the character area into six banks
; Area 1 (2Kbyte, characters $00 - $7F) $0000 - $07FF -use CBSR0
; Area 2 (2Kbyte, characters $80 - $FF) $0800 - $0FFF -use CBSR1
; Area 3 (1Kbyte, characters $00 - $3F) $1000 - $13FF -use CBSR2
; Area 4 (1Kbyte, characters $40 - $7F) $1400 - $17FF -use CBSR3
; Area 5 (1Kbyte, characters $80 - $BF) $1800 - $1BFF -use CBSR4
; Area 6 (1Kbyte, characters $C0 - $FF) $1C00 - $1FFF -use CBSR5

MMC3_IRQTIMER EQU $E000 ;Disable Timer

```

```

MMC3_IRQCOUNTER      EQU $E001 ;Enable Timer
MMC3_INTERVALTIMER   EQU $C000
MMC3_INTERVALCOUNTER EQU $C001
MSSR      EQU $A000 ;Scroll Selection Register - 0=horizontal,1=vertical
           EQU $A001 ;Write Protect Register - bit 7 selects write
           ;protect (1) ;and write enable (0)
MBRS      EQU $8000 ;Bank Register Select Register - used to select which
           ;bank register is to be written to through MBRV
MBRV      EQU $8001 ;Bank Select Value Register - Actual value to be
           ;written to the MMC3 register pointed to by MBRS
CBSR0     EQU $00 ;Character Bank Select Register 0
CBSR1     EQU $01 ;Character Bank Select Register 1
CBSR2     EQU $02 ;Character Bank Select Register 2
CBSR3     EQU $03 ;Character Bank Select Register 3
CBSR4     EQU $04 ;Character Bank Select Register 4
CBSR5     EQU $05 ;Character Bank Select Register 5
PBSR0     EQU $06 ;Program Bank Select Register 0
PBSR1     EQU $07 ;Program Bank Select Register 1

;
;
;      *** MMC-1 ***
;
;Multi-Memory Controller 1 (MMC1) Equates
;MMC registers are written to serially by storing values consecutively to
;address to fill their defined bits, e.g. 5 sta's are necessary to defined
;MMC_R0.
MMC1      EQU $8000 ;Store something with bit 7 high ($80 recommend
; Store High Bit ($80) to clear MMC (initializes the SP Registers and
; sets MMC_R0 to default values
MMC_R0    EQU $8000 ;was $9FFF
; 0 - Horizontal Scroll (set to 0) or Set to Vertical Scroll (set to 1)
; 1 - Single screen mode (set to 0) or 4 screen mode (set to 1)
; 2 - Resident Bank, 0=$8000-$BFFF and 1=$C000-$FFFF
; 3 - Single Bank Size, 0=32K and 1=16K
; 4 - Character ROM bank unit 4/8, 0=8K bank and 1=4K bank
MMC_R1    EQU $A000 ;was BFFF
; If MMC_R0 Bit 4 is 0 (8K banks selected):
; 0 - Ignored
; 1 to 4 - 8K Character Bank # to have active (Character Bank Code, PPU $0
; If MMC_R0 Bit 4 is 1 (4K banks selected):
; 0 to 4 - 4K Character Bank # to have active (Character Bank Code, PPU $0
MMC_R2    EQU $C000 ;was DFFF
; If MMC_R0 Bit 4 is 1 (4K banks selected):
; 0 to 4 - 4K Character Bank # to have active (Character Bank Code, PPU $1
; If MMC_R0 Bit 4 is 0 (8K banks selected), this register is ignored
MMC_R3    EQU $E000 ;was FFFF
; If MMC_R0 Bit 3 is 1 (16K program banks selected):
; 0 to 3 - 16K program bank to to switch in (program bank designation code
; 4 - Program Bank call (set to 0) and RAMbank designation call (set
BANK0     EQU %00000000
BANK1     EQU %00000001
BANK2     EQU %00000010
BANK3     EQU %00000011
BANK4     EQU %00000100
BANK5     EQU %00000101
BANK6     EQU %00000110
BANK7     EQU %00000111
BANK8     EQU %00001000
BANK9     EQU %00001001
BANK10    EQU %00001010
BANK11    EQU %00001011
BANK12    EQU %00001100
BANK13    EQU %00001101
BANK14    EQU %00001110
BANK15    EQU %00001111

```

```

;Rom Emulator Softswitch to enable (bit 0 is off) or disable (bit 0 is on) MMC1
ROM_SOFTSWITCH EQU $7FFF

```

•DDT Memory Equates

```

BANK1 EQU $0000
BANK2 EQU $1000
VIDPAGE1 EQU $2000
COLMAP1 EQU $23C0
COLMAP1b EQU $23E0
VIDPAGE2H EQU $2400
COLMAP2H EQU $27C0
COLMAP2Hb EQU $27E0
VIDPAGE2V EQU $2800
COLMAP2V EQU $2BC0
VIDPAGEEX EQU $2C00
COLMAPEX EQU $2FC0

```

;Second half of color areas for page 1

;Second half of color areas for page 2H

```

COLORS EQU $3f00
CHCOL1.1 EQU $3F01
CHCOL1.2 EQU $3F02
CHCOL1.3 EQU $3F03
;unused EQU $3f04
CHCOL2.1 EQU $3F05
CHCOL2.2 EQU $3F06
CHCOL2.3 EQU $3F07
;unused EQU $3f08
CHCOL3.1 EQU $3F09
CHCOL3.2 EQU $3F0a
CHCOL3.3 EQU $3F0b
;unused EQU $3f0c
CHCOL4.1 EQU $3F0d
CHCOL4.2 EQU $3F0e
L4.3 EQU $3F0f
LOR EQU $3F10

```

```

SPCOL1.1 EQU $3F11
SPCOL1.2 EQU $3F12
SPCOL1.3 EQU $3F13
;unused EQU $3f14
SPCOL2.1 EQU $3F15
SPCOL2.2 EQU $3F16
SPCOL2.3 EQU $3F17
;unused EQU $3f18
SPCOL3.1 EQU $3F19
SPCOL3.2 EQU $3F1A
SPCOL3.3 EQU $3F1B
;unused EQU $3f1c
SPCOL4.1 EQU $3F1D
SPCOL4.2 EQU $3F1E
SPCOL4.3 EQU $3F1F

```

;Hardware Registers

```

PCR1 EQU $2000 ;PPU Control Register 1
; 0 - MSB horizontal scroll +256
; 1 - MSB vertical scroll +240
; 2 - Auto increment mode 0-> +1, 1-> +32
; 3 - Sprite Char Mem Add 0->$0000, 1->$1000
; 4 - Background Char Mem Add 0->$0000, 1->$1000
; 5 - Sixteen Pixel tall sprite mode
; 6 - unused set to 0
; 7 - Vertical blank NMI enable
EQU $2001 ;PPU Control Register 2
; 0 - Monochrome mode
; 1 - Border width extend bit
; 2 - unused set to 1
; 3 - Background enable

```

```

;
; 4 - Sprite enable
; 5 - RED color intensity adjust
; 6 - GREEN color intensity adjust
; 7 - BLUE color intensity adjust
;
; EQU $2002          ;PPU Status Register
; 0-5 unused
; 6 - collision detection bit(sp0 and background)
; 7 - vertical blank occurred bit
PPCR EQU $2003          ;Sprite Register Destination Address - Where in the page
PDAT EQU $2004          ;Sprite Parameter Data Port - Appears to be what page to
PSOR EQU $2005          ;PPU Scroll Register
PMAR EQU $2006          ;PPU DMA Address Register/Display Address
PMDR EQU $2007          ;PPU DMA Data Register

V1R1 EQU $4000          ;Voice 1 Duty Cycle
; 0-3 - Amplitude
; 4-5 - Envelope
; 6-7 - Duty Cycle
V1R2 EQU $4001          ;Frequency Sweep Controls
; 7 - sweep enable
; 4-6 - don't care
; 3 - direction
; 0-2 - Sweep delay, or cut-off value
V1R3 EQU $4002          ;Voice 1 Frequency lower 8 bits
V1R4 EQU $4003          ;Voice 1 Frequency upper 3 bits/envelope length
; 0-2 - Upper 3 bits of frequency
; 3-7 - length code for envelope
V2R1 EQU $4004
; 0-3 - Amplitude
; 4-5 - Envelope
; 6-7 - Duty Cycle
V2R2 EQU $4005          ;Frequency Sweep Controls
; 7 - sweep enable
; 4-6 - don't care
; 3 - direction
; 0-2 - Sweep delay, or cut-off value
V2R3 EQU $4006          ;Voice 2 Frequency lower 8 bits
V2R4 EQU $4007          ;Voice 2 Frequency upper 3 bits/envelope length
; 0-2 - Upper 3 bits of frequency
; 3-7 - length code for envelope
V3R1 EQU $4008
; 0-3 - Amplitude
; 4-5 - Envelope          ;different from other voices
; 6-7 - Duty Cycle
V3R3 EQU $400A          ;Voice 3 Frequency lower 8 bits
V3R4 EQU $400B          ;Voice 3 Frequency upper 3 bits/envelope length
; 0-2 - Upper 3 bits of frequency
; 3-7 - length code for envelope
V4R1 EQU $400C
V4R2 EQU $400D          ;controls for voice 4 noise channel
V4R3 EQU $400E
V4R4 EQU $400F
V5R1 EQU $4010          ;Sample Sound Playback Rate,loop,IRQ enable
; 0-3 - playback rate
; 6 - loop mode
; 7 - IRQ enable
V5R2 EQU $4011          ;Sample Sound Shift Register
; *NOTE: Bit 7 = Ignored, Bit 0 = Up/Down Counter, Bit
V5R3 EQU $4012          ;Base Address of Sample Sound {$C000 + (V5R3*$40)}
V5R4 EQU $4013          ;length of sample Mult of $10 Bytes

EQU $4014          ;Sprite DMA Start Address (high) - What Page to Xfer

PAD1 EQU $4016          ;game controller 1
PAD2 EQU $4017          ;game controller 2
; Must be initialized to $40-$ff during initialization to stop

```

spurious IRQ's.

```
VMER EQU $4015 ;Sound Master Enable Register
.    0 - Voice 1 enable
    1 - Voice 2 enable
    2 - Voice 3 enable
;    3 - Voice 4 enable
;    4 - Sample Sound Voice enable
```

; Sprite Equates and RAM Addresses

```
SPRITE EQU $700 ;Page #7 contains all the Sprite Handle Informat
SPRITE0 EQU SPRITE
SPRITE0_Y EQU SPRITE
SPRITE0_N EQU SPRITE+1
SPRITE0_C EQU SPRITE+2
SPRITE0_X EQU SPRITE+3
SPRITE1 EQU SPRITE+4
SPRITE1_Y EQU SPRITE+4
SPRITE1_N EQU SPRITE+5
SPRITE1_C EQU SPRITE+6
SPRITE1_X EQU SPRITE+7
SPRITE2 EQU SPRITE+8
SPRITE2_Y EQU SPRITE+8
SPRITE2_N EQU SPRITE+9
SPRITE2_C EQU SPRITE+10
SPRITE2_X EQU SPRITE+11
SPRITE3 EQU SPRITE+12
SPRITE3_Y EQU SPRITE+12
SPRITE3_N EQU SPRITE+13
SPRITE3_C EQU SPRITE+14
SPRITE3_X EQU SPRITE+15
SPRITE4 EQU SPRITE+16
SPRITE4_Y EQU SPRITE+16
SPRITE4_N EQU SPRITE+17
SPRITE4_C EQU SPRITE+18
SPRITE4_X EQU SPRITE+19
SPRITE5 EQU SPRITE+20
SPRITE5_Y EQU SPRITE+20
SPRITE5_N EQU SPRITE+21
SPRITE5_C EQU SPRITE+22
SPRITE5_X EQU SPRITE+23
SPRITE6 EQU SPRITE+24
SPRITE6_Y EQU SPRITE+24
SPRITE6_N EQU SPRITE+25
SPRITE6_C EQU SPRITE+26
SPRITE6_X EQU SPRITE+27
SPRITE7 EQU SPRITE+28
SPRITE7_Y EQU SPRITE+28
SPRITE7_N EQU SPRITE+29
SPRITE7_C EQU SPRITE+30
SPRITE7_X EQU SPRITE+31
SPRITE8 EQU SPRITE+32
SPRITE8_Y EQU SPRITE+32
SPRITE8_N EQU SPRITE+33
SPRITE8_C EQU SPRITE+34
SPRITE8_X EQU SPRITE+35
SPRITE9 EQU SPRITE+36
SPRITE9_Y EQU SPRITE+36
SPRITE9_N EQU SPRITE+37
SPRITE9_C EQU SPRITE+38
SPRITE9_X EQU SPRITE+39
SPRITE10 EQU SPRITE+40
SPRITE10_Y EQU SPRITE+40
SPRITE10_N EQU SPRITE+41
SPRITE10_C EQU SPRITE+42
SPRITE10_X EQU SPRITE+43
```

SPRITE11	EQU SPRITE+44
SPRITE11_Y	EQU SPRITE+44
SPRITE11_N	EQU SPRITE+45
SPRITE11_C	EQU SPRITE+46
SPRITE11_X	EQU SPRITE+47
SPRITE12	EQU SPRITE+48
SPRITE12_Y	EQU SPRITE+48
SPRITE12_N	EQU SPRITE+49
SPRITE12_C	EQU SPRITE+50
SPRITE12_X	EQU SPRITE+51
SPRITE13	EQU SPRITE+52
SPRITE13_Y	EQU SPRITE+52
SPRITE13_N	EQU SPRITE+53
SPRITE13_C	EQU SPRITE+54
SPRITE13_X	EQU SPRITE+55
SPRITE14	EQU SPRITE+56
SPRITE14_Y	EQU SPRITE+56
SPRITE14_N	EQU SPRITE+57
SPRITE14_C	EQU SPRITE+58
SPRITE14_X	EQU SPRITE+59
SPRITE15	EQU SPRITE+60
SPRITE15_Y	EQU SPRITE+60
SPRITE15_N	EQU SPRITE+61
SPRITE15_C	EQU SPRITE+62
SPRITE15_X	EQU SPRITE+63
SPRITE16	EQU SPRITE+64
SPRITE16_Y	EQU SPRITE+64
SPRITE16_N	EQU SPRITE+65
SPRITE16_C	EQU SPRITE+66
SPRITE16_X	EQU SPRITE+67
SPRITE17	EQU SPRITE+68
SPRITE17_Y	EQU SPRITE+68
SPRITE17_N	EQU SPRITE+69
SPRITE17_C	EQU SPRITE+70
SPRITE17_X	EQU SPRITE+71
SPRITE18	EQU SPRITE+72
SPRITE18_Y	EQU SPRITE+72
SPRITE18_N	EQU SPRITE+73
SPRITE18_C	EQU SPRITE+74
SPRITE18_X	EQU SPRITE+75
SPRITE19	EQU SPRITE+76
SPRITE19_Y	EQU SPRITE+76
SPRITE19_N	EQU SPRITE+77
SPRITE19_C	EQU SPRITE+78
SPRITE19_X	EQU SPRITE+79
SPRITE20	EQU SPRITE+80
SPRITE20_Y	EQU SPRITE+80
SPRITE20_N	EQU SPRITE+81
SPRITE20_C	EQU SPRITE+82
SPRITE20_X	EQU SPRITE+83
SPRITE21	EQU SPRITE+84
SPRITE21_Y	EQU SPRITE+84
SPRITE21_N	EQU SPRITE+85
SPRITE21_C	EQU SPRITE+86
SPRITE21_X	EQU SPRITE+87
SPRITE22	EQU SPRITE+88
SPRITE22_Y	EQU SPRITE+88
SPRITE22_N	EQU SPRITE+89
SPRITE22_C	EQU SPRITE+90
SPRITE22_X	EQU SPRITE+91
SPRITE23	EQU SPRITE+92
SPRITE23_Y	EQU SPRITE+92
SPRITE23_N	EQU SPRITE+93
SPRITE23_C	EQU SPRITE+94
SPRITE23_X	EQU SPRITE+95
SPRITE24	EQU SPRITE+96

SPRITE24_Y	EQU SPRITE+96
SPRITE24_N	EQU SPRITE+97
SPRITE24_C	EQU SPRITE+98
SPRITE24_X	EQU SPRITE+99
TE25	EQU SPRITE+100
TE25_Y	EQU SPRITE+100
SPRITE25_N	EQU SPRITE+101
SPRITE25_C	EQU SPRITE+102
SPRITE25_X	EQU SPRITE+103
SPRITE26	EQU SPRITE+104
SPRITE26_Y	EQU SPRITE+104
SPRITE26_N	EQU SPRITE+105
SPRITE26_C	EQU SPRITE+106
SPRITE26_X	EQU SPRITE+107
SPRITE27	EQU SPRITE+108
SPRITE27_Y	EQU SPRITE+108
SPRITE27_N	EQU SPRITE+109
SPRITE27_C	EQU SPRITE+110
SPRITE27_X	EQU SPRITE+111
SPRITE28	EQU SPRITE+112
SPRITE28_Y	EQU SPRITE+112
SPRITE28_N	EQU SPRITE+113
SPRITE28_C	EQU SPRITE+114
SPRITE28_X	EQU SPRITE+115
SPRITE29	EQU SPRITE+116
SPRITE29_Y	EQU SPRITE+116
SPRITE29_N	EQU SPRITE+117
SPRITE29_C	EQU SPRITE+118
SPRITE29_X	EQU SPRITE+119
SPRITE30	EQU SPRITE+120
SPRITE30_Y	EQU SPRITE+120
SPRITE30_N	EQU SPRITE+121
TE30_C	EQU SPRITE+122
TE30_X	EQU SPRITE+123
SPRITE31	EQU SPRITE+124
SPRITE31_Y	EQU SPRITE+124
SPRITE31_N	EQU SPRITE+125
SPRITE31_C	EQU SPRITE+126
SPRITE31_X	EQU SPRITE+127
SPRITE32	EQU SPRITE+128
SPRITE32_Y	EQU SPRITE+128
SPRITE32_N	EQU SPRITE+129
SPRITE32_C	EQU SPRITE+130
SPRITE32_X	EQU SPRITE+131
SPRITE33	EQU SPRITE+132
SPRITE33_Y	EQU SPRITE+132
SPRITE33_N	EQU SPRITE+133
SPRITE33_C	EQU SPRITE+134
SPRITE33_X	EQU SPRITE+135
SPRITE34	EQU SPRITE+136
SPRITE34_Y	EQU SPRITE+136
SPRITE34_N	EQU SPRITE+137
SPRITE34_C	EQU SPRITE+138
SPRITE34_X	EQU SPRITE+139
SPRITE35	EQU SPRITE+140
SPRITE35_Y	EQU SPRITE+140
SPRITE35_N	EQU SPRITE+141
SPRITE35_C	EQU SPRITE+142
SPRITE35_X	EQU SPRITE+143
SPRITE36	EQU SPRITE+144
TE36_Y	EQU SPRITE+144
TE36_N	EQU SPRITE+145
SPRITE36_C	EQU SPRITE+146
SPRITE36_X	EQU SPRITE+147
SPRITE37	EQU SPRITE+148
SPRITE37_Y	EQU SPRITE+148

SPRITE37_N	EQU SPRITE+149
SPRITE37_C	EQU SPRITE+150
SPRITE37_X	EQU SPRITE+151
SPRITE38	EQU SPRITE+152
SPRITE38_Y	EQU SPRITE+152
SPRITE38_N	EQU SPRITE+153
SPRITE38_C	EQU SPRITE+154
SPRITE38_X	EQU SPRITE+155
SPRITE39	EQU SPRITE+156
SPRITE39_Y	EQU SPRITE+156
SPRITE39_N	EQU SPRITE+157
SPRITE39_C	EQU SPRITE+158
SPRITE39_X	EQU SPRITE+159
SPRITE40	EQU SPRITE+160
SPRITE40_Y	EQU SPRITE+160
SPRITE40_N	EQU SPRITE+161
SPRITE40_C	EQU SPRITE+162
SPRITE40_X	EQU SPRITE+163
SPRITE41	EQU SPRITE+164
SPRITE41_Y	EQU SPRITE+164
SPRITE41_N	EQU SPRITE+165
SPRITE41_C	EQU SPRITE+166
SPRITE41_X	EQU SPRITE+167
SPRITE42	EQU SPRITE+168
SPRITE42_Y	EQU SPRITE+168
SPRITE42_N	EQU SPRITE+169
SPRITE42_C	EQU SPRITE+170
SPRITE42_X	EQU SPRITE+171
SPRITE43	EQU SPRITE+172
SPRITE43_Y	EQU SPRITE+172
SPRITE43_N	EQU SPRITE+173
SPRITE43_C	EQU SPRITE+174
SPRITE43_X	EQU SPRITE+175
SPRITE44	EQU SPRITE+176
SPRITE44_Y	EQU SPRITE+176
SPRITE44_N	EQU SPRITE+177
SPRITE44_C	EQU SPRITE+178
SPRITE44_X	EQU SPRITE+179
SPRITE45	EQU SPRITE+180
SPRITE45_Y	EQU SPRITE+180
SPRITE45_N	EQU SPRITE+181
SPRITE45_C	EQU SPRITE+182
SPRITE45_X	EQU SPRITE+183
SPRITE46	EQU SPRITE+184
SPRITE46_Y	EQU SPRITE+184
SPRITE46_N	EQU SPRITE+185
SPRITE46_C	EQU SPRITE+186
SPRITE46_X	EQU SPRITE+187
SPRITE47	EQU SPRITE+188
SPRITE47_Y	EQU SPRITE+188
SPRITE47_N	EQU SPRITE+189
SPRITE47_C	EQU SPRITE+190
SPRITE47_X	EQU SPRITE+191
SPRITE48	EQU SPRITE+192
SPRITE48_Y	EQU SPRITE+192
SPRITE48_N	EQU SPRITE+193
SPRITE48_C	EQU SPRITE+194
SPRITE48_X	EQU SPRITE+195
SPRITE49	EQU SPRITE+196
SPRITE49_Y	EQU SPRITE+196
SPRITE49_N	EQU SPRITE+197
SPRITE49_C	EQU SPRITE+198
SPRITE49_X	EQU SPRITE+199
SPRITE50	EQU SPRITE+200
SPRITE50_Y	EQU SPRITE+200
SPRITE50_N	EQU SPRITE+201

SPRITE50_C	EQU SPRITE+202
SPRITE50_X	EQU SPRITE+203
SPRITE51	EQU SPRITE+204
SPRITE51_Y	EQU SPRITE+204
SPRITE51_N	EQU SPRITE+205
SPRITE51_C	EQU SPRITE+206
SPRITE51_X	EQU SPRITE+207
SPRITE52	EQU SPRITE+208
SPRITE52_Y	EQU SPRITE+208
SPRITE52_N	EQU SPRITE+209
SPRITE52_C	EQU SPRITE+210
SPRITE52_X	EQU SPRITE+211
SPRITE53	EQU SPRITE+212
SPRITE53_Y	EQU SPRITE+212
SPRITE53_N	EQU SPRITE+213
SPRITE53_C	EQU SPRITE+214
SPRITE53_X	EQU SPRITE+215
SPRITE54	EQU SPRITE+216
SPRITE54_Y	EQU SPRITE+216
SPRITE54_N	EQU SPRITE+217
SPRITE54_C	EQU SPRITE+218
SPRITE54_X	EQU SPRITE+219
SPRITE55	EQU SPRITE+220
SPRITE55_Y	EQU SPRITE+220
SPRITE55_N	EQU SPRITE+221
SPRITE55_C	EQU SPRITE+222
SPRITE55_X	EQU SPRITE+223
SPRITE56	EQU SPRITE+224
SPRITE56_Y	EQU SPRITE+224
SPRITE56_N	EQU SPRITE+225
SPRITE56_C	EQU SPRITE+226
SPRITE56_X	EQU SPRITE+227
SPRITE57	EQU SPRITE+228
SPRITE57_Y	EQU SPRITE+228
SPRITE57_N	EQU SPRITE+229
SPRITE57_C	EQU SPRITE+230
SPRITE57_X	EQU SPRITE+231
SPRITE58	EQU SPRITE+232
SPRITE58_Y	EQU SPRITE+232
SPRITE58_N	EQU SPRITE+233
SPRITE58_C	EQU SPRITE+234
SPRITE58_X	EQU SPRITE+235
SPRITE59	EQU SPRITE+236
SPRITE59_Y	EQU SPRITE+236
SPRITE59_N	EQU SPRITE+237
SPRITE59_C	EQU SPRITE+238
SPRITE59_X	EQU SPRITE+239
SPRITE60	EQU SPRITE+240
SPRITE60_Y	EQU SPRITE+240
SPRITE60_N	EQU SPRITE+241
SPRITE60_C	EQU SPRITE+242
SPRITE60_X	EQU SPRITE+243
SPRITE61	EQU SPRITE+244
SPRITE61_Y	EQU SPRITE+244
SPRITE61_N	EQU SPRITE+245
SPRITE61_C	EQU SPRITE+246
SPRITE61_X	EQU SPRITE+247
SPRITE62	EQU SPRITE+248
SPRITE62_Y	EQU SPRITE+248
SPRITE62_N	EQU SPRITE+249
SPRITE62_C	EQU SPRITE+250
SPRITE62_X	EQU SPRITE+251
SPRITE63	EQU SPRITE+252
SPRITE63_Y	EQU SPRITE+252
SPRITE63_N	EQU SPRITE+253
SPRITE63_C	EQU SPRITE+254

SPRITE63_X

EQU SPRITE+255

```

;File: EQUATES.NES
; Richard L. Seaborne's NES equate file by Richard L. Seaborne
; Copyright 1990 ALL RIGHTS RESERVED
; Date: 1/23/90

```

(Controller) Equates (what each bit in a Controller Byte is)

```

PADRIGHT EQU %00000001
PADLEFT EQU %00000010
PADDOWN EQU %00000100
PADUP EQU %00001000
PADSTART EQU %00010000
PADSELECT EQU %00100000
PADBUTTONB EQU %01000000
PADBUTTONA EQU %10000000

```

*** RAMBO ***

; RAMBO (NAMCO 109) Memory Controller Equates

; PROGRAM BANKING:

```

; The RAMBO splits the program area into four 8Kbyte banks
;   Area 1 (8Kbyte) $8000 - $9FFF - set through PBSR0
;   Area 2 (8Kbyte) $A000 - $BFFF - set through PBSR1
;   Area 3 (8Kbyte) $C000 - $DFFF - set through PBSR2
;   Area 4 (8Kbyte) $E000 - $FFFF - fixed at highest bank

```

; CHARACTER BANKING:

```

; The RAMBO splits the character area into six banks
;   Area 1 (2Kbyte, characters $00 - $7F) $0000 - $07FF -use CBSR0
;   Area 2 (2Kbyte, characters $80 - $FF) $0800 - $0FFF -use CBSR1
;   Area 3 (1Kbyte, characters $00 - $3F) $1000 - $13FF -use CBSR2
;   Area 4 (1Kbyte, characters $40 - $7F) $1400 - $17FF -use CBSR3
;   Area 5 (1Kbyte, characters $80 - $BF) $1800 - $1BFF -use CBSR4
;   Area 6 (1Kbyte, characters $C0 - $FF) $1C00 - $1FFF -use CBSR5

```

```

BPTR EQU $8000 ;Bits 0-3 (lower nibble):
;   Bank Pointer Register - used to select which
;   bank register is to be written to through Rn
;Bit 4: NOT USED, KEEP 0 for future compatibility
;Bit 5: Character Map Mode for Bank0
;   0=2Kbyte CHR Bank (128 stamps/bank)
;   1=1Kbyte CHR Bank (64 stamps/bank)
;Bit 6: Program Map Mode
;   Reverses Window-0 and Window-2 (set to 0 for
;   MMC-3 compatibility)
;Bit 7: Character Segment
;   0=Normal NES CHR bank selection with PCR1 ($2000)
;   1=Reverses meaning of Bits 3 and 4 of PCR1 ($2000)

Rn EQU $8001 ;Bits 1-7: (Char Map Mode 0)
;   Register Number Register - Actual value to be
;   written to the RAMBO register pointed to by BPTR
;Bits 0-7: (Char Map Mode 1)
;   Register Number Register - Actual value to be
;   written to the RAMBO register pointed to by BPTR
;Bits 0-5: (Program Selection)
;   Register Number Register - Actual value to be
;   written to the RAMBO register pointed to by BPTR

SCRL EQU $A000 ;Bit 0: Scroll Control Register
;   0=horizontal
;   1=vertical
;Bit 1: 48-Pin Rambo ONLY
;   0=No Playfield RAM
;   1=Playfield RAM on Cartridge

;NOTE: Write-Protect Register of MMC3 ($A001) is not supported by RAMBO
TLAT EQU $C000 ;Timer Count Latch - this is the value loaded into
;the IRQ timer after every IRQ

```